Apple Performance Tools

Session 905

Apple Performance Tools

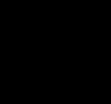
Robert Bowdidge
Developer Tools Group, Apple Computer

Why Worry About Performance?

- Performance is a selling point
- Performance problems are (mostly) invisible
- You cannot worry about only your application
    - Users will run multiple applications
    - Does your application play well with others?
- How do you find performance problems in your application?

What Will You Learn Today?

- Apple has tools to help
  - Tools can measure, identify possible causes
  - You will see quick examples of tool use
  - Developer CD tools
  - Processor performance tools
- See framework talks for specific, concrete advice
- Explore tool use on your own



Causes of Poor Performance

- Excessive use of memory
- Executing too much code
- Waiting for other processes or devices
    - Excessive disk accesses
    - Excessive drawing or redrawing
- Most problems cause excessive memory use
- Most memory use problems cause swapping

What Tools Exist?

	Memory Use	Execution Behavior	Resource Use
Monitor	top heap vmmap	top Thread Viewer	fs_usage sc_usage QuartzDebug
Analyze	MallocDebug ObjectAlloc	Sampler Shikari MONster	Sampler OpenGL Profiler

Example Program: SimPhysics

- Realistic example, with real inefficiencies
- Originally for Java, quickly ported to Cocoa
- Visualizes electrical fields between particles

Finding Performance Problems

- How do we notice performance problems?
  - Measurements!
  - Egregious problems
  - Expectations about complexity, resource use
  - Comparison with past performance
- First step: run **top** command line tool
  - Provides details about system state
  - CPU, memory usage of tasks, paging rate
  - Watch other relevant processes (Window Server)
- Most relevant: CPU usage, swapping, RPRVT

Check Memory Use Over Time

- How can you identify trends in memory use?
  - Call **top** repeatedly, save results to file
- Command-line scripts can be your friend:
  - Permit customized data collection
  - Run remotely, run without affecting system
  - Run even when window server will not respond

Causes of Excessive Memory Use

- Too much memory allocated
  - Large structures scattered across memory
  - Unneeded caching of data in memory
  - Leaking unneeded memory
- Excessive memory use leads to disk accesses

Finding Causes of Memory Use

- MallocDebug: track source of current allocations
  - Relates allocations to call graphs
  - Allocations since marked point
  - Leak detection
  - Pointer problems
- ObjectAlloc: how many objects exist?
  - Shows number of objects of each class
  - Shows changes over time

MallocDebug: Where Is Memory Allocated?

- Shows all allocations, grouped by allocation site
  - Browse call graph to find uses at site
  - Examine contents of memory
  - Identify calls responsible for large allocations
- Only shows currently allocated objects

MallocDebug: Are We Leaking Any Memory?

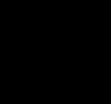
- What memory are we allocating, then forgetting?
  - Leaks waste memory, fatal for long-running applications
  - Force scattering of memory that is used
- MallocDebug can detect unreferenced blocks
  - Searches all memory for pointers to blocks
  - Unreferenced blocks must be leaked
  - Only root of leaked structure shown!

MallocDebug: Finding Pointer Bugs

- Some memory bugs can be subtle, intermittent
  - Referencing freed memory causes values to change
  - Overrunning/underrunning buffers trash memory
- MallocDebug "helps" detect such problems
  - Encourages badly behaved program to crash
  - Overwrites freed memory, adds guard words
  - Assorted other warning messages to console

ObjectAlloc: Analyzing Object Use

- ObjectAlloc shows how many objects exist
  - Current, max, total created over life of program
  - Shows totals, bar graph, backtraces
- Great for summary, noting trends
  - Did you expect 100 FooBars?
- Works for CoreFoundation, ObjC objects
  - Backtraces for Objective-C retains and releases
  - Also shows malloc blocks



CPU Usage: What Are The Problems?

- Cause of excessive CPU time
  - Executing unnecessary code
  - Expensive algorithm
  - Unexpected calls to expensive code
  - Checking for events by polling, not blocking
- General approach: find most costly routines
  - Improvements will make largest difference

Sampler: Where Are the Hot Spots?

- To improve speed, you need to find costly calls
- Sampler finds what code is executing
  - Stops program at interval, records call stack
  - Shows only functions seen at sampling time
  - Presents call graph with sample counts
- If you need better detail, check out gprof

Causes of Poor Resource Use

- Inappropriate disk accesses
  - Who says files have to be on local machine?
- Multiple processes may divide work
  - Drawing partially done by window server
  - Waiting for responses

Are We Using the Disk Badly?

- What's happening at startup to keep us slow?
- Maybe we are doing too many disk accesses
  - Accesses at wrong time gives slow appearance
- Use **fs_usage** to watch disk operations
  - Must be run as root or with **sudo**
- Use Sampler to see where we called routines

Are We Drawing Too Much?

- Drawing directly affects performance
  - Drawing consumes CPU time, memory
  - Idle windows consume memory
  - Drawing eventually done by window server
- Use QuartzDebug to understand drawing
  - Immediately perceive redundant drawing
  - Learn about offscreen windows

How Did We Do With SimPhysics?

	Plain	Intensity Arrows	Color Graph
Before	34.0 fps	2.1 fps	2.3 fps
After	19.9 fps	15.2 fps	6.9 fps

- Memory use: 2.5 MB before, 1.1 MB after
- Measured on 600MHz iBook with lots of memory

Other Tools to Check Out Yourself

- Heap memory use
  - **heap, leaks , malloc_history**
- Process state:
  - **sample**: Command-line version of Sampler
  - **time**: overall time for specific application
- Resources
  - **io_stat**: disk, network performance (Jaguar only)
  - **sc_usage**: which system calls are used?

Hints About Tool Use

- All tools work on any binary
  - No need to instrument or recompile
- CFM binaries generated by CodeWarrior
  - Use inline traceback tables for symbol info
  - Set macro to use system's malloc library

How to Learn More About Tools

- Part of Mac OS X Developer Tools CD
- Documentation for applications
  - Graphical tools: documentation in application
  - Command-line tools: **man** pages
  - **/Developer/Documentation/ReleaseNotes**
- Inside Mac OS X books
  - “Inside Mac OS X: Performance”
  - “Inside Mac OS X: System Overview”

Conclusion

- Tune your application to make best impression
- Aim to reduce memory use . . . in all ways
- Compare performance between builds
- Watch Window Manager and other servers
- Create great applications for Mac OS X

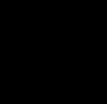
Computer Hardware Understanding Development Tools

"CHUD" Tools

Eric Miller and Nathan Slingerland
Architecture Performance Group, Apple Computer

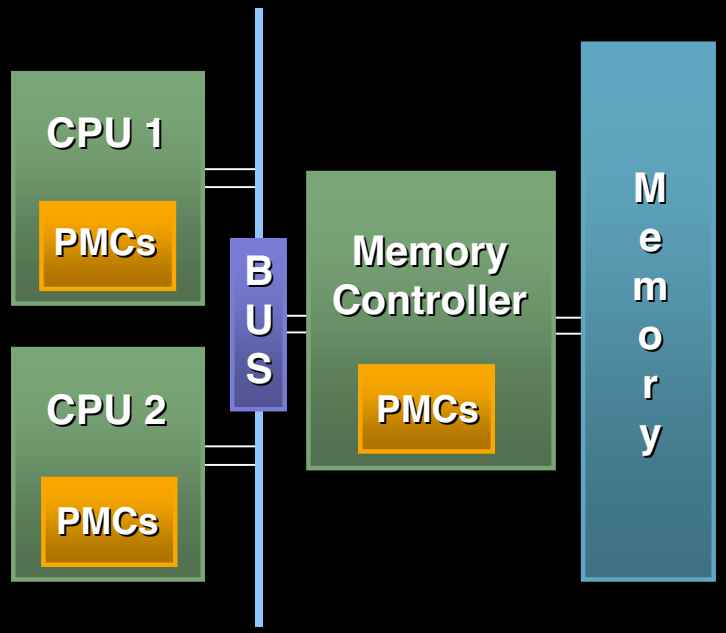
What Can You Do With CHUD Tools?

- Find out what the hardware is doing
- Find critical code segments and tune that code
- Look at system-wide behavior



What Is the Hardware Doing?

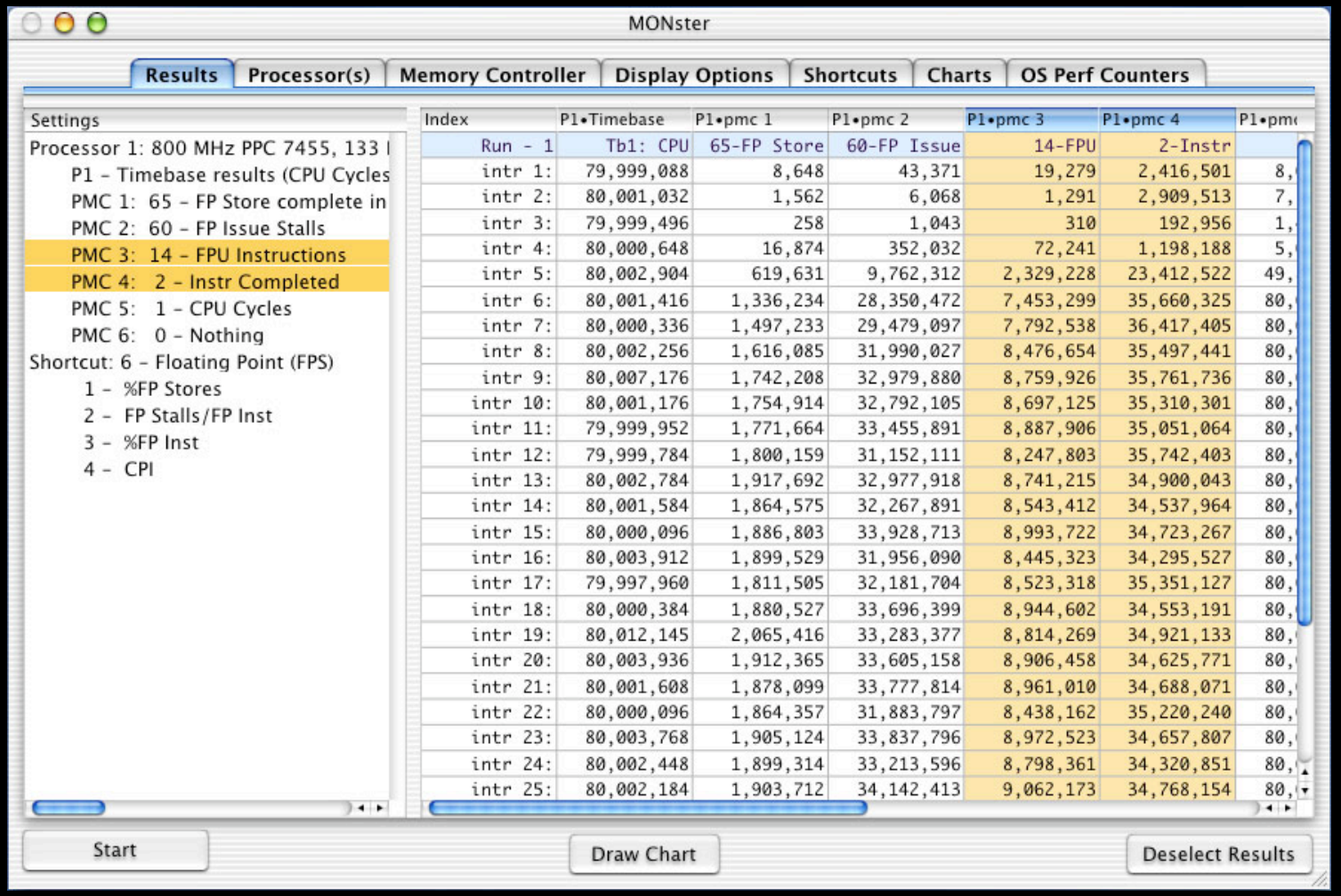
- Macintosh hardware has built-in Performance Monitor Counters (PMCs)
- Count "performance events"
  - Instructions
  - Cache misses
  - Execution stalls
- Use CHUD tools to ask the PMCs what is going on



MONster

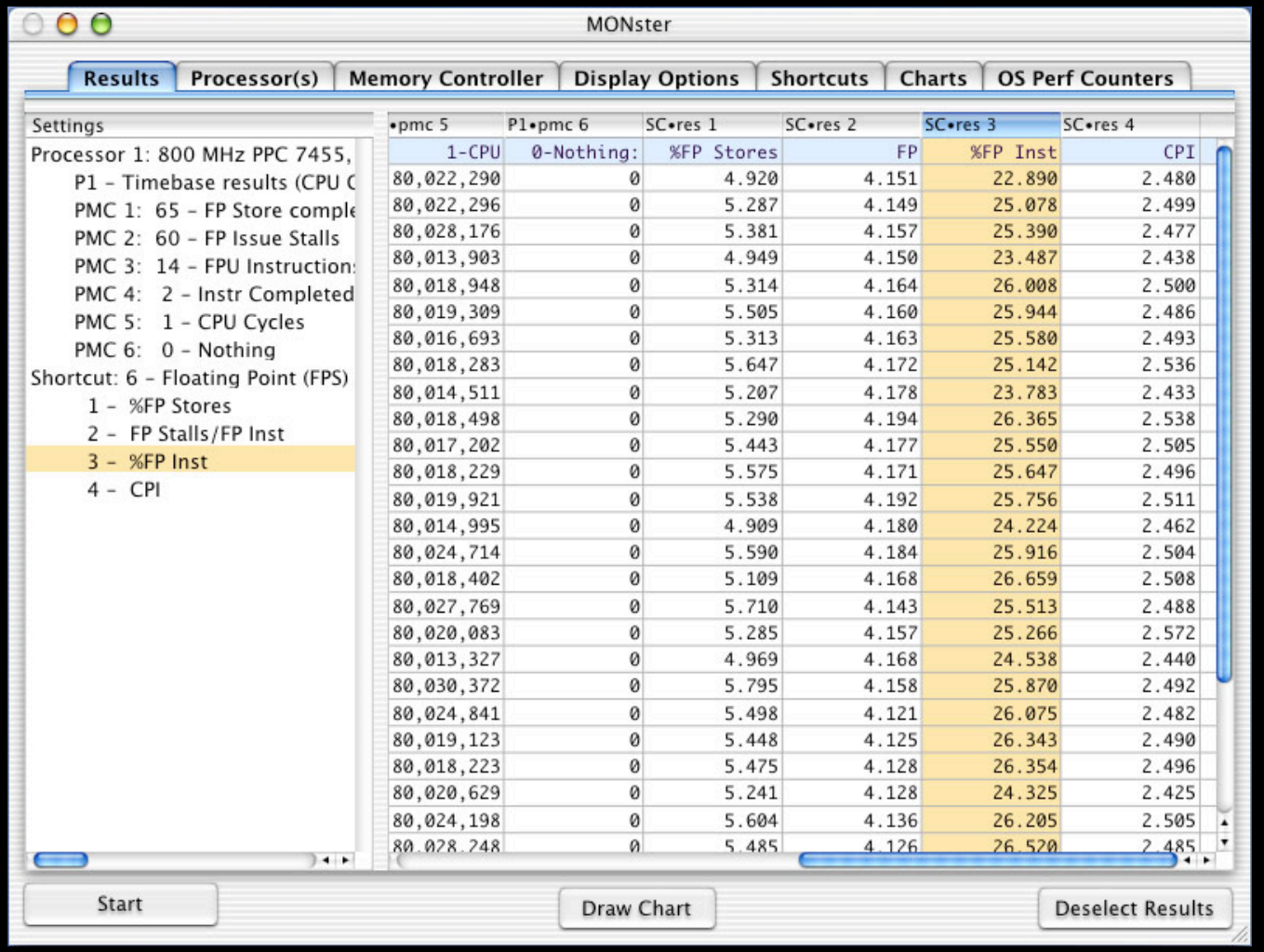
- Configure Performance Monitor Counters
- Collect PMC data based on:
  - Hotkey, timed intervals, or event counts
- Compute performance metrics from PMC data
  - Memory bandwidth (bytes/sec)
  - Cycles per instruction (CPI)
  - Define your own metrics
- View results in tabular or chart form

MONster

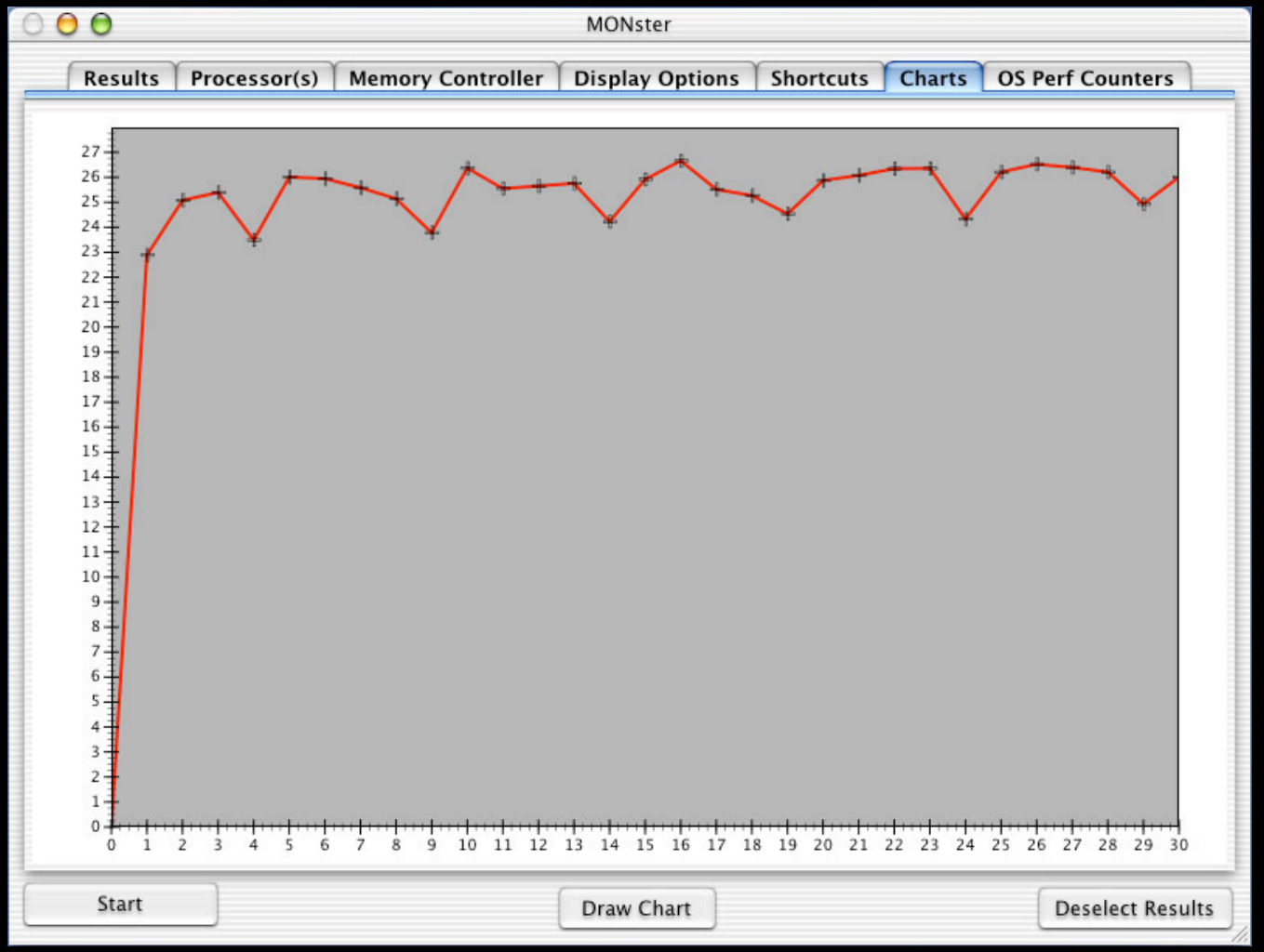


Index	P1•Timebase	P1•pmc 1	P1•pmc 2	P1•pmc 3	P1•pmc 4	P1•pmc
Run - 1	Tb1: CPU	65-FP Store	60-FP Issue	14-FPU	2-Instr	
intr 1:	79,999,088	8,648	43,371	19,279	2,416,501	8,
intr 2:	80,001,032	1,562	6,068	1,291	2,909,513	7,
intr 3:	79,999,496	258	1,043	310	192,956	1,
intr 4:	80,000,648	16,874	352,032	72,241	1,198,188	5,
intr 5:	80,002,904	619,631	9,762,312	2,329,228	23,412,522	49,
intr 6:	80,001,416	1,336,234	28,350,472	7,453,299	35,660,325	80,
intr 7:	80,000,336	1,497,233	29,479,097	7,792,538	36,417,405	80,
intr 8:	80,002,256	1,616,085	31,990,027	8,476,654	35,497,441	80,
intr 9:	80,007,176	1,742,208	32,979,880	8,759,926	35,761,736	80,
intr 10:	80,001,176	1,754,914	32,792,105	8,697,125	35,310,301	80,
intr 11:	79,999,952	1,771,664	33,455,891	8,887,906	35,051,064	80,
intr 12:	79,999,784	1,800,159	31,152,111	8,247,803	35,742,403	80,
intr 13:	80,002,784	1,917,692	32,977,918	8,741,215	34,900,043	80,
intr 14:	80,001,584	1,864,575	32,267,891	8,543,412	34,537,964	80,
intr 15:	80,000,096	1,886,803	33,928,713	8,993,722	34,723,267	80,
intr 16:	80,003,912	1,899,529	31,956,090	8,445,323	34,295,527	80,
intr 17:	79,997,960	1,811,505	32,181,704	8,523,318	35,351,127	80,
intr 18:	80,000,384	1,880,527	33,696,399	8,944,602	34,553,191	80,
intr 19:	80,012,145	2,065,416	33,283,377	8,814,269	34,921,133	80,
intr 20:	80,003,936	1,912,365	33,605,158	8,906,458	34,625,771	80,
intr 21:	80,001,608	1,878,099	33,777,814	8,961,010	34,688,071	80,
intr 22:	80,000,096	1,864,357	31,883,797	8,438,162	35,220,240	80,
intr 23:	80,003,768	1,905,124	33,837,796	8,972,523	34,657,807	80,
intr 24:	80,002,448	1,899,314	33,213,596	8,798,361	34,320,851	80,
intr 25:	80,002,184	1,903,712	34,142,413	9,062,173	34,768,154	80,

MONster


MONster
Results
Processor(s)
Memory Controller
Display Options
Shortcuts
Charts
OS Perf Counters
Settings
Processor 1: 800 MHz PPC 7455,
P1 - Timebase results (CPU C
PMC 1: 65 - FP Store comple
PMC 2: 60 - FP Issue Stalls
PMC 3: 14 - FPU Instruction
PMC 4: 2 - Instr Completed
PMC 5: 1 - CPU Cycles
PMC 6: 0 - Nothing
Shortcut: 6 - Floating Point (FPS)
1 - %FP Stores
2 - FP Stalls/FP Inst
3 - %FP Inst
4 - CPI
•pmc 5 | P1•pmc 6 | SC•res 1 | SC•res 2 | SC•res 3 | SC•res 4
1-CPU | 0-Nothing: | %FP Stores | FP | %FP Inst | CPI
80,022,290 | 0 | 4.920 | 4.151 | 22.890 | 2.480
80,022,296 | 0 | 5.287 | 4.149 | 25.078 | 2.499
80,028,176 | 0 | 5.381 | 4.157 | 25.390 | 2.477
80,013,903 | 0 | 4.949 | 4.150 | 23.487 | 2.438
80,018,948 | 0 | 5.314 | 4.164 | 26.008 | 2.500
80,019,309 | 0 | 5.505 | 4.160 | 25.944 | 2.486
80,016,693 | 0 | 5.313 | 4.163 | 25.580 | 2.493
80,018,283 | 0 | 5.647 | 4.172 | 25.142 | 2.536
80,014,511 | 0 | 5.207 | 4.178 | 23.783 | 2.433
80,018,498 | 0 | 5.290 | 4.194 | 26.365 | 2.538
80,017,202 | 0 | 5.443 | 4.177 | 25.550 | 2.505
80,018,229 | 0 | 5.575 | 4.171 | 25.647 | 2.496
80,019,921 | 0 | 5.538 | 4.192 | 25.756 | 2.511
80,014,995 | 0 | 4.909 | 4.180 | 24.224 | 2.462
80,024,714 | 0 | 5.590 | 4.184 | 25.916 | 2.504
80,018,402 | 0 | 5.109 | 4.168 | 26.659 | 2.508
80,027,769 | 0 | 5.710 | 4.143 | 25.513 | 2.488
80,020,083 | 0 | 5.285 | 4.157 | 25.266 | 2.572
80,013,327 | 0 | 4.969 | 4.168 | 24.538 | 2.440
80,030,372 | 0 | 5.795 | 4.158 | 25.870 | 2.492
80,024,841 | 0 | 5.498 | 4.121 | 26.075 | 2.482
80,019,123 | 0 | 5.448 | 4.125 | 26.343 | 2.490
80,018,223 | 0 | 5.475 | 4.128 | 26.354 | 2.496
80,020,629 | 0 | 5.241 | 4.128 | 24.325 | 2.425
80,024,198 | 0 | 5.604 | 4.136 | 26.205 | 2.505
80,028,248 | 0 | 5.485 | 4.126 | 26.520 | 2.485
Start
Draw Chart
Deselect Results

MONster

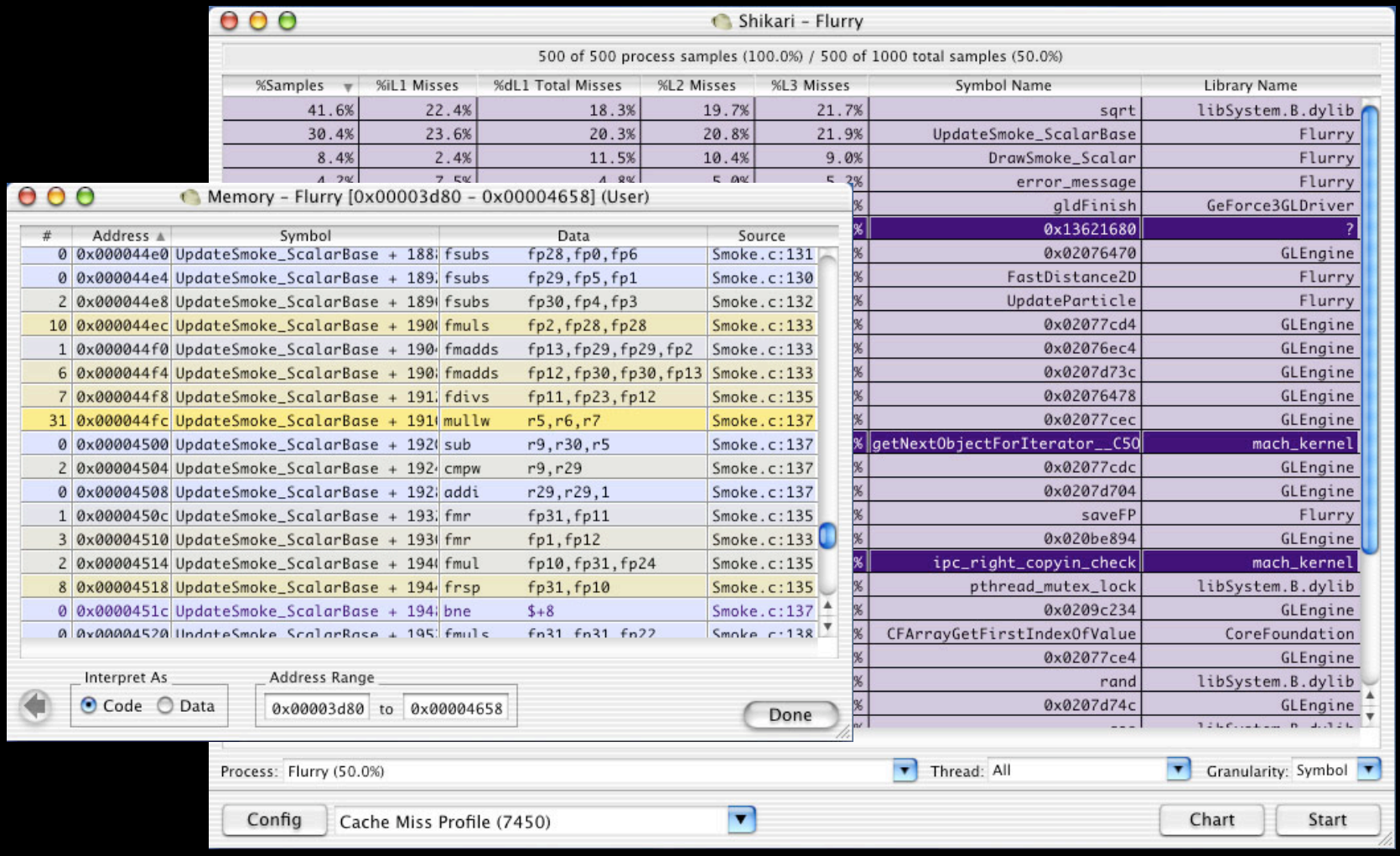

MONster
Results
Processor(s)
Memory Controller
Display Options
Shortcuts
Charts
OS Perf Counters
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Start
Draw Chart
Deselect Results

Shikari

shi·ka·ri (sh-kär, -kr) *n. pl.* **shi·ka·ris,** a big game hunter; a professional hunter or guide

- Sample what is running on the system
  - Use event counts or timed intervals
  - System wide, process, or thread scope
- Relate performance events to your source code
  - Performance “hot spots”
  - Annotated disassembly
- Performance event histograms

Shikari Demo



CHUD.framework

- Write your own performance analysis tool
  - Control Performance Monitor Counters
  - Access Special Purpose Registers (SPRs)
  - Collect information about system hardware
    - CPU, Memory Controller, Bus
- Instrument your source code around critical sections

CHUD.framework Example

```
#include <CHUD/chud.h>
…
chudInitialize();

printf(" CPU: PPC %d, %d MHz, Bus: %d
MHz\n",
    chudProcessorType(),chudProcessorMHz(),
    chudBusFreq()/1000000);


…
chudSetCounterEvent(chudCPU1Dev, PMC_1, 1);
chudSetCounterEvent(chudCPU1Dev, PMC_2, 2);

chudClearCounters();
chudStartCounters();

your_important_function();

chudStopCounters();
…

CHUDreport();
```

Setup and Gather Data

```
void CHUDreport( void ) {
/* get counter data */
pmcCounts=chudGetEventCounts(chudCPU1Dev,&ct);

/* output counter data */
 for (i=0; i<ct; i++) {
   ev = chudGetCounterEvent(chudCPU1Dev, i);
   evstr = chudGetEventString(i, ev);
   printf("\tPMC: %d\t\t%f\t%s\n",
       i+1,pmcCounts[i],evstr)); }}
```

Report Results

Other CHUD Tools

- Amber—instruction tracer
- Acid—filters traces for perf statistics
- SimG4—cycle accurate G4 (7400) simulator
- Reggie—examine and modify SPRs

Where to Get CHUD Tools

- Available on the web:
  http://developer.apple.com/tools/debuggers.html

  (Then click on **CHUD Tools** to download)

- Report any issues to:
  chud-tools-feedback@group.apple.com

For More Information

- "Mac OS X: System Overview"
  "Mac OS X: Performance"
  on disk:
  /Developer/Documentation/Essentials/
  orderable in print from the web:
  http://developer.apple.com/techpubs/
- Apple Developer Connection tools page
  http://developer.apple.com/tools/
- Bug Reporting
  http://developer.apple.com/bugreporter/

Roadmap

906 Developing for Performance: Understand system performance concepts	Hall 2 **Fri., 9:00am**
907 Compiler Developments at Apple: See the tools to optimize your application	Room J **Fri., 10:30am**
FF015 Development Tools: Make your thoughts known	Room J1 **Fri., 3:30pm**
909 Debugging in Mac OS X: Learn about gdb and debugging techniques	Hall 2 **Fri., 5:00pm**

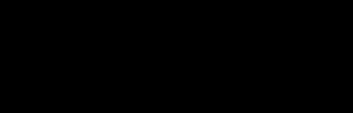
Who to Contact

Godfrey DiGiorgi
Technology Manager, Development Tools
ramarren@apple.com

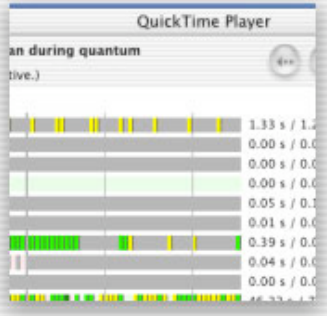
Development Tools Engineering Feedback
macosx-tools-feedback@group.apple.com

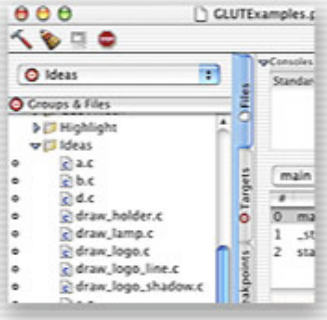
CHUD-Tools Engineering Feedback
chud-tools-feedback@group.apple.com

http://developer.apple.com/wwdc2002/urls.html



Q&A





Godfrey DiGiorgi
Technology Manager, Development Tools
ramarren@apple.com

http://developer.apple.com/wwdc2002/urls.html


WWDC2002


WWDC2002


WWDC2002